Name : Rupesh Dhirwani	**Class/Roll No. :** D16AD - 10	**Grade :**

Title of Experiment:
Design and Implement a fully connected Deep Neural Network.

Objective of Experiment :
The objective is to design and implement a deep neural network for classification and to test it using appropriate dataset. To build an optimized neural network with higher test accuracy by using Regularization, dropout and early stop etc.

Outcome of Experiment :
To understand how to build a neural network using keras in python and to learn the optimization of neural networks by using different methods like regularization, dropout and early stop etc. To also learn to optimize the learning rate.

Problem Statement :
Develop a deep neural network model for a classification application that can accurately classify input data into distinct categories. The model will consist of at least 2 hidden layers and will use appropriate learning algorithms, output functions, and loss functions.

Description / Theory :

Adam Optimizer:
Adaptive Moment Estimation The Adam optimizer, short for Adaptive Moment Estimation, is an optimization algorithm widely used in training neural networks. It combines the benefits of two other popular optimization algorithms, namely RMSProp and momentum-based gradient descent. Adam adapts the learning rates of individual parameters and maintains moving averages of both the gradient and the squared gradient of the loss function. This adaptive nature makes it well-suited for various deep learning tasks and helps overcome some of the limitations of other optimization algorithms.

L2 Regularization (Weight Decay):
L2 regularization is applied to the dense layers of the neural network. This technique involves adding a penalty term to the loss function proportional to the square of the magnitude of weights. By doing so, the model is discouraged from learning overly complex features that might lead to overfitting. It helps prevent large weights and encourages the network to focus on important features.

Deep Learning/Odd Sem 2023-23/Experiment 2c

Dropout:
Dropout is implemented using dropout layers after each dense (hidden) layer. Dropout randomly deactivates a fraction of neurons during each training iteration. This regularization technique prevents neurons from relying too heavily on specific input features and encourages the model to learn more robust representations. Dropout helps reduce overfitting by preventing the network from becoming overly specialized to the training data.

Early Stopping:
The EarlyStopping callback is used during training to monitor the validation loss. If the validation loss stops improving for a specified number of epochs (patience), training is stopped early. Early stopping helps prevent overfitting by preventing the model from memorizing noise in the training data.

Algorithm/ Pseudo Code / Flowchart (whichever is applicable):

Algorithm: Building and Training a Deep Neural Network for Classification

Input:
- Training dataset (X_train, y_train)
- Validation dataset (X_val, y_val)
- Test dataset (X_test, y_test)
- Number of hidden layers (at least 2)
- Number of neurons in each hidden layer
- Learning algorithm
- Output function
- Loss function
- Hyperparameters (learning rate, batch size, epochs, dropout rate, etc.)

Output:
- Trained neural network model.
- Evaluation metrics on test dataset.

1. Preprocessing:
- Normalize or scale the input features if required.

2. Initialize the Neural Network Model:
- Initialize an empty neural network model.
- Add an input layer with the appropriate input shape.

3. Add Hidden Layers:
- For each hidden layer (at least 2):
- Add a fully connected (dense) layer with the specified number of neurons.

- Apply an appropriate activation function (e.g., ReLU) to the layer's output.

4. Add Output Layer:
- Add an output layer with the number of neurons equal to the number of classes.
- Apply the chosen output function (e.g., softmax) to the output layer.

5. Compile the Model:
- Compile the model using the chosen learning algorithm, loss function, and relevant metrics.
- Specify the optimizer (e.g. Adam) and learning rate.

6. Train the Model:
- Train the model using the training dataset:
- Use the fit() method, specifying the training data, labels, batch size, and number of epochs.
- Implement early stopping using a callback to prevent overfitting.
- Use dropout layers during training for regularization.

7. Evaluate on Validation Set:
- Evaluate the trained model's performance on the validation dataset:
- Use the evaluate() method to compute validation loss and metrics.
- Track validation accuracy, precision, recall, etc.

8. Hyperparameter Tuning:
- Adjust hyperparameters (e.g., number of neurons, learning rate, dropout rate) based on validation results.

9. Test the Final Model:
- Evaluate the trained model on the test dataset:
- Use the evaluate() method to compute test loss and metrics.
- Calculate test accuracy, precision, recall, etc.

10. Results and Conclusion:
- Summarize the model's performance and impact of different architectural choices.
- Present evaluation metrics and insights on model generalization.

Program :
Before Optimization Used Breast Cancer Dataset:

```
import numpy as np
import tensorflow as tf
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from tensorflow import keras
```

Deep Learning/Odd Sem 2023-23/Experiment 2c

```
from tensorflow.keras import layers
# Load the Breast Cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target
# Convert labels to one-hot encoding
y_one_hot = tf.keras.utils.to_categorical(y, num_classes=2)
# Split the data into training and testing sets
X_train,   X_test,   y_train,   y_test   =   train_test_split(X,
y_one_hot,

test_size=0.2, random_state=42)
# Build the neural network model
model = keras.Sequential([
    layers.Input(shape=(30,)),
    layers.Dense(128, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(2, activation='softmax')
])
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
              metrics=['accuracy'])
# Train the model
batch_size = 32
epochs = 50
model.fit(X_train,        y_train,        batch_size=batch_size,
epochs=epochs,
          validation_split=0.2)
# Evaluate the model on the test data
test_loss, test_accuracy = model.evaluate(X_test, y_test)
```

```
print(f"Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.4f}")
```

```
Epoch 48/50
12/12 [==============================] - 0s 5ms/step - loss: 2.1723 - accuracy: 0.7885 - val_loss: 5.0262 - val_accuracy: 0.5165
Epoch 49/50
12/12 [==============================] - 0s 5ms/step - loss: 2.9354 - accuracy: 0.7555 - val_loss: 0.9567 - val_accuracy: 0.8791
Epoch 50/50
12/12 [==============================] - 0s 5ms/step - loss: 0.9198 - accuracy: 0.8984 - val_loss: 0.9009 - val_accuracy: 0.8462
4/4 [==============================] - 0s 2ms/step - loss: 0.4569 - accuracy: 0.9211
Test Loss: 0.4569, Test Accuracy: 0.9211
```

Accuracy is 91%

After Optimization Used Breast Cancer Dataset:

```
import numpy as np
import tensorflow as tf
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.callbacks import EarlyStopping
# Load the Breast Cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target
# Convert labels to one-hot encoding
y_one_hot = tf.keras.utils.to_categorical(y, num_classes=2)
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_one_hot,

test_size=0.2, random_state=42)
# Build the neural network model
model = keras.Sequential([
    layers.Input(shape=(30,)),
    layers.Dense(128, activation='relu',
```

```
kernel_regularizer=keras.regularizers.l2(0.01)),
    layers.Dropout(0.5),  # Adding dropout with a rate of 0.5
    layers.Dense(64, activation='relu',

kernel_regularizer=keras.regularizers.l2(0.01)),
    layers.Dropout(0.5),  # Adding dropout with a rate of 0.5
    layers.Dense(2, activation='softmax')
])
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
              metrics=['accuracy'])
# Implement early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=10,
                               restore_best_weights=True)
# Train the model with early stopping
batch_size = 32
epochs = 100
model.fit(X_train,         y_train,         batch_size=batch_size,
epochs=epochs,
          validation_split=0.2, callbacks=[early_stopping])
# Evaluate the model on the test data
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test     Loss:     {test_loss:.4f},     Test     Accuracy:
{test_accuracy:.4f}")
```

```
Epoch 21/100
12/12 [==============================] - 0s 5ms/step - loss: 2.1516 - accuracy: 0.6868 - val_loss: 1.0009 - val_accuracy: 0.8352
Epoch 22/100
12/12 [==============================] - 0s 5ms/step - loss: 1.9379 - accuracy: 0.6813 - val_loss: 1.0238 - val_accuracy: 0.6923
Epoch 23/100
12/12 [==============================] - 0s 5ms/step - loss: 1.9561 - accuracy: 0.6896 - val_loss: 0.9533 - val_accuracy: 0.8901
Epoch 24/100
12/12 [==============================] - 0s 5ms/step - loss: 1.9768 - accuracy: 0.6813 - val_loss: 1.0047 - val_accuracy: 0.6813
Epoch 25/100
12/12 [==============================] - 0s 7ms/step - loss: 1.5163 - accuracy: 0.7473 - val_loss: 0.9693 - val_accuracy: 0.6923
4/4 [==============================] - 0s 3ms/step - loss: 0.7226 - accuracy: 0.9298
Test Loss: 0.7226, Test Accuracy: 0.9298
```

Accuracy is 93%

Results and Discussions :

The model was built using Adam Optimizer and used breast cancer dataset for classification. Before using optimization methods the test accuracy was 85% on 50 Epochs. After using optimization methods like L2 Regularization, Dropout and early stopping the test accuracy was improved to 93%. In this experiment we learned to implement a deep neural network using keras and learned to optimize the neural networks using different methods on a dataset.